EXOPTIM: design of Experiment that are Optimally Planned and Time-varying for Identification of Models

Matlab/Scala Toolbox

Boite à outils en Matlab pour un problème d'optimisation d'expériences

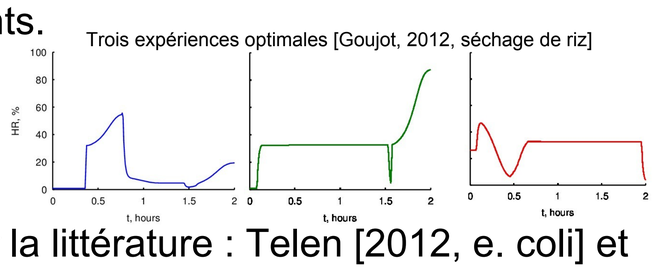
Finalité: obtenir de meilleurs résultats d'identification de paramètres méconnus de modèles (de simulation), en utilisant ce modèle pour prédire des résultats d'expériences.

Sous-titre

EXOPTIM, Une nouvelle boîte à outils pour expériences optimales

- Nécessite du matériel (un process), et un modèle non-linéaire le simulant, qui dépend explicitement de l'état au temps t et non-linéairement en les paramètres.
- Contexte : on a changé le produit, il faut mettre à jour le modèle à coût matériel minimal : Design of Optimal Experiments.

Trois expériences optimales [Goujot, 2012, séchage de riz]



validé sur la littérature : Telen [2012, e. coli] et Pronzato [2010, pharmacologie].

- validé sur pilote Goujot [2012, séchage de riz], autre validation en cours: DPPH+Isoeugenol en réacteur.
- Limite : 10^4 simulations.

Les autres optimisations d'expérience publiées, classées par type de modèle :

- Linéaire en paramètres: Logiciels courants, cas courant.
- Non linéaires en paramètres : 10^9 simulations [Pazman 92], [Vila 2007].
- Localement linéarisé en paramètres:
  - 10^6 simulations [RodriguezFernandez 2006]
  - logiciel propriétaire [Psenterprise]
  - ignore corrélations [Mathieu 2011]
  - optimise que temps de prélèvement [Ucinski 2005].

Détails technique du code :

- 1.2 Mo de Matlab (6.6 Mo cumulés dans CVS).
- Pas de test de régression
- Interface stable, seuls certains détails changent.
- Utilisé à Toulouse aussi (LGC).
- Goulot d'étranglement refait en Scala, l'accélérant de 5000 %. (cf exemple : produit matrice-vecteur en Scala ; profiler créé pour mes besoins)
- Détachable de la licence Matlab. (mcc)

Accélération avec le langage Scala du coeur, choisi car parallélisable et fait bien les petits produits matrice-vecteur, et virtuellement parallélisable

Fichier maClasse.scala :

```
package fr.inra ;
import scalaSci.EJML.StaticMathsEJML._
import scalaSci.EJML._
import scalaSci.EJML.Mat
class maClasse(val m_ : Array[Array[Double]])
val m=new Mat(m_)
def maMethode (val v_ : Array[Int]):Array[Array[Double]] = {
val v=new Mat(Array(v_))
Val monProduit=m*v ; Syntaxe facilitant traduction Matlab-Scala
Array(monProduit.toArray,(v+v).toArray)
}
```

Compilation :

```
scalac -d . -classpath $JARS maClass.scala
jar cf monScala.jar fr/
```

Dans Matlab :

```
Jarlist=cat(...
' monScala.jar',...
' scala-library.jar',...
' scalalab-2.9.1.jar') ;
unix(['cat',Jarlist,'>>',which('classpath.txt')]) ;
out=fr.inra.maClasse([1 0 0 ;1 1 1]).maMethode([0 1]) ;ni Struct ni Cell ni String
monProduit=reshape(out{1},2,2) ;Accepte plusieurs sorties
```

Désolidarisation du jeton Matlab, avec le « compilateur » Matlab sans se séparer de Scala.

```
mcc -mv maFonction.m -a monScala.jar -a scala-library.jar -a scalalab-2.9.1.jar : un « compilateur » qui ralentit négligeablement, mais compatible Scala.
```

```
}
object Profile { a recopier dans chaque classe scala
 private[this] val t0 = System.currentTimeMillis private[this] accélère Scala
 private[this] val NbLine = 2000// maximal number of lines
 private[this] var Count = new Array[Long] (10 * NbLine)
 private[this] var tCumulated = new Array[Long] (10 * NbLine)// sum of times of executions.
 private[this] var DisplayWhen : Long = 1000
 def storeClock(noline:Int) = { Count(noline) += 1; tCumulated(noline) += System.currentTimeMillis - t0; } inséré par preprocesseur à chaque ligne de méthode
 def output:Unit = {
  var CumulatedCount : Long = 0 ; for ( i <- 0 until 10 * NbLine) { CumulatedCount += Count(i) }
  if (CumulatedCount > DisplayWhen) { var lasti=i; i = -1;
    while (i < 10 * NbLine-1) { i+=1 ; if (Count(i)>0) {
      if (Count(lasti)==Count(i) && i>0)
          println(""+((tCumulated(i)-tCumulated(lasti)))+"="+Count(lasti)+"x"+((0.+tCumulated(i)-tCumulated(lasti))/Count(lasti))+" ms (base "+tCumulated(lasti)+") at planif_sequentielle line "+(lasti/10.));
       else
          println("?="+Count(i)+"x? µs (base "+tCumulated(lasti)+") at planif_sequentielle line "+(lasti/10.)); lasti=i; }
   } while bien plus rapide que for en Scala
    DisplayWhen += 2 * DisplayWhen ; println("Sleeping for "+DisplayWhen+" more executions of Profile.storeClock.")
  }
 }
```

Références

Goujot, D., Meyer, X.~M., & Courtois, F. (2012). Identification of a rice drying model with an improved sequential optimal design of experiments. *Journal of Process Control*, 22(1), 95--107.

Telen, D., Logist, F., Van Derlinden, E., Tack, I., & Van Impe, J. (2012). *Optimal experiment design for dynamic bioprocesses: A multi-objective approach*. Chemical Engineering Science, 78(0), 82-97.

Lomel, F., Mathieu, S., Machefer, L., Falk, J.-M., & Commenge, S. (2011). Stratégie expérimentale de détermination des paramètres d'un modèle de procédé en vue du choix d'une technologie intensifiée. In *SFGP* Lille.

Pázman, A. & Pronzato, L. (1992). Nonlinear experimental design based on the distribution of estimators. *Journal of Statistical Planning and Inference*, 33(3), 385--402.

Uciński, D. (2005). *Optimal measurement methods for distributed parameter system identification*. CRC Press.

Vila, J.-P. & Gauchi, J.-P. (2007). Optimal designs based on exact confidence regions for parameter estimation of a nonlinear regression model. *Journal of Statistical Planning and Inference*, 137(9), 2935--2953.

Walter, E. & Pronzato, L. (2010). *Identification of Parametric Models from Experimental Data*. London: Springer.

Daniel.goujot@agroparistech.fr

Daniel GOUJOT[a,b], Xuan-Mi MEYER[c], Francis COURTOIS[b,a]…

UMR1145 Génial — LGC

[a]INRA/[b]AgroParisTech - Centre de Massy — [c]CNRS/INPT/UPS 5503

1, av des Olympiades, F-91300 Massy

www.inra.fr

www.agroparistech.fr